Journal of The Institute of Electronics and Information Engineers Vol.61, NO.12, December 2024 https://doi.org/10.5573/ieie.2024.61.12.32
ISSN 2287-5026(Print) / ISSN 2288-159X(Online)

논문 2024-61-12-5

Questa를 이용한 Code Coverage 검증 방법

(Code Coverage Verification Method using Questa)

엄 유 진*, 양 희 훈*, 심 용 기**, 배 득 우****, 오 민 식****, 유 호 영***

(Yujin Eom, Heehun Yang, Yonggi Sim, Deukwoo Bae, Minsik Oh, and Hoyoung Yoo©)

요 약

반도체 공정 기술의 발전으로 인해 증가한 집적도는 디지털 회로 설계 단계에서 강력한 검증 기술의 필요성을 높이고 있다. 다양한 설계 검증 방법 중, 본 논문에서는 RTL 코드 검증 과정에서 Questa를 활용한 코드 커버리지 검증 절차를 소개한다. 먼저 Questa 툴을 기반으로 코드 커버리지 검증 방법을 설명하고, 이를 활용한 두 가지 예제인 카운터와 FIFO 예제를 통해 설계 오류를 확인하고 세부 수정 절차를 자세히 다룬다. 이를 통해 코드 커버리지 검증이 하드웨어 설계의 신뢰성과 무결성을 높이는 데 있어 효율적인 도구임을 확인할 수 있다. 끝으로, 코드 커버리지 검증 방법이 디지털 회로의 완성도와 무결성을 향상시키며, 이는 항공, 우주, 국방, 자동차 등 다양한 산업 분야에서 시스템의 안정성과 신뢰성을 보장하는 데 기여할 수 있음을 제시한다.

Abstract

Advances in semiconductor process technology have increased integration density, which has made strong verification techniques more important in digital circuit design. This paper introduces a method for code coverage verification using Questa during the RTL code verification process. It first explains how code coverage works with the Questa tool and explores two examples: a counter and a FIFO. These examples show how to find and fix design errors in detail. This demonstrates that code coverage is an effective tool for improving the reliability and integrity of hardware design. Lastly, the paper highlights how code coverage can enhance the completeness and integrity of digital circuits, contributing to the stability and reliability of systems in industries like aerospace, defense, and automotive.

Keywords : Code coverage, Questa, DO-254, Verification

Ⅰ. 서 론

현대 디지털 회로 기술은 무어의 법칙에 따라 지속해서 발전하고 있다. 이러한 반도체 공정 기술의 진보는 같은 면적의 칩에 더 많은 회로를 집적할 수 있도록 하였다. 그러나 이러한 집적도의 증가는 회로의 복잡도를 대폭 상승시켰고, 이에 따라 회로의 안정성을 확보하기 위해서는 설계 단계보다 검증 단계에 더 많은 시간과 자원이 소모되게 되었다[1, 2]. 특히, FPGA(Field Programmable Gate Array)나 ASIC(Application Specific Integrated Circuit)을 포함한 디지털 회로를 설계할 때에는 RTL(Register Transfer Level) 단계에서의 검증 과정이 매우 중요하다. 초기 단계에서의 검증 없이는 로직 합성 및 배치와 배선 과정이 시작되기

*학생회원, **비회원, ***정회원, 충남대학교 전자공학과(Department of Electronics Engineering, Chungnam National University)
****비회원, 한국항공우주산업(주)(KOREA AEROSPAC업 INDUSTRIES, LTD.)
© Corresponding Author(E-mail : hyyoo@cnu.ac.kr)
※ This research was supported by National Research Foundation of Korea(NRF) funded by the Korea government(MSIT) (RS-2021-II210779 (50%), 2020M3H2A1078119) and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2022R1A5A8026986, 2022-0-01170 (PIM 반도체설계 연구센터)).
Received ; May 10, 2024 Revised ; May 22, 2024 Accepted ; September 14, 2024

표 1. Code coverage 유형
Table 1. Code coverage types.

Coverage type	Description
Statement	각 문장이 실행되었는지를 보이는 coverage
Branch	If, case 구문에서 분기가 실행되었는지를 보이는 coverage
Condition	If 구문에서 모든 조건이 실행되었는지를 보이는 coverage
Toggle	모든 신호가 전환 가능한 모든 값 사이에서 전환되는지를 보이는 coverage
Expression	Assign 구문에서 모든 표현이 실행되었는지를 보이는 coverage
FSM	FSM에서 모든 상태와 천이가 수행되었는지를 보이는 coverage

전에 발생할 수 있는 다양한 문제들을 효과적으로 파악하고 해결하는 것이 어렵다[3].

이런 배경에서, RTL 코드의 검증을 위해서는 그 안정성과 신뢰성을 보장할 수 있는 효과적인 방법이 필요하게 되었다. 그중에서도 code coverage 검증 방법은 다른 방법들에 비해 비교적 간단하며, 설계 초기 단계에서 발생할 수 있는 다양한 설계 오류와 테스트 케이스의 부족을 신속하게 발견하는 데 큰 도움을 준다. 전통적으로 code coverage는 소프트웨어 검증 분야에서 소프트웨어의 설계 품질을 평가하고 향상시키기 위한 중요한 도구로 널리 사용되어 왔다. 최근에는 하드웨어 검증 분야에도 이 방법이 도입되어 디지털 회로의 초기 설계 단계에서 효율적인 검증을 위한 중요한 지표로 활용되고 있다. 특히 하드웨어 설계에서는 시뮬레이션 및 검증 기반의 접근법이 주로 사용되며, 이 과정에서 코드의 실행 여부를 중심으로 분석해 회로의 검증을 진행한다[4, 5]. 이 방법은 특정 코드 부분이 실행되었는지를 직관적으로 파악하고, 개발 초기 단계에서 소요 시간을 최소화하며 오류를 신속하게 찾아내는 데 큰 이점을 제공한다. 이 과정을 통해 발견된 오류가 설계 자체의 결함인지, 충분한 테스트 조건의 부재로 인한 문제인지를 명확하게 파악할 수 있으며 필요한 경우 RTL 코드를 수정하거나 테스트 벤치를 보완할 수 있다. 예를 들어, [6]에서는 마이크로프로세서의 설계 과정에서 code coverage를 이용한 검증이 8%의 오류를 쉽게 식별해 낼 수 있었다는 결과를 보여주고 있다. 이러한 결과는 code coverage 검증이 디지털 회로 설계 과정에서 매우 유용한 도구로 활용될 수 있음을 강조한다.

본 논문에서는 이러한 배경을 바탕으로, RTL 초기 단계에서 활용 가능한 효과적인 검증 절차인 code coverage를 counter와 FIFO 설계 사례를 통해 소개한다. Mentor사의 Questa 2020.3 버전을 사용하여, code coverage 세부 검증 방법을 단계별로 설명한다. 이후 섹션에서는 두 가지 예제에 대한 code coverage 검증 결과를 분석하고, 이를 바탕으로 수정 절차를 제시한다. 끝으로, 결론 부분에서 연구의 결과와 그 의의를 정리하고, 향후 연구 방향에 대해 논의한다.

```
1: always @(posedge Clk or posedge Rst)
2:   if(Rst)
3:     CurState <= ST_IDLE;
4:   else
5:     CurState <= NextState;
```

그림 1. Statement coverage 예제
Fig. 1. Statement coverage example.

Ⅱ. 배 경

다양한 EDA (Electronic Design Automation) 공급업체들은 자체적으로 code coverage를 지원하는 다양한 도구들을 제공하고 있다. 이 중에서 Mentor 사의 Questa와 Synopsys 사의 VCS는 시뮬레이션 도구로서 code coverage 결과를 제공해 설계 및 검증 과정에서 중요한 역할을 한다. 본 논문에서는 특히 Mentor 사의 Questa를 중심으로 code coverage를 분석하는 방법을 자세히 다룬다. 표 1과 같이 Questa는 code coverage를 여섯 가지 주요 유형으로 구분하여 제공하며, 각각은 설계 검증 과정에서 특정한 목적을 수행한다.

1. Statement Coverage

Statement coverage는 코드 내 각 문장이 최소 한 번 이상 실행되었는지를 확인하는 유형이다. 코드의 어떤 부분이 실행되었는지, 숨겨진 오류가 없는지 확인할 수 있다. 이는 코드의 각 부분이 테스트되었는지 보장한다. 예를 들어, 그림 1에서 3, 5번째 줄의 실행 여부는 statement coverage에서 커버해야 하는 목록으로써 나타난다. 이러한 목록의 충족 여부는 간단하면서도 정보를 신속하게 제공하기 때문에 초기 검증 단계에서 매우 유용하다.

2. Branch Coverage

Branch coverage는 조건문 (if/case 구문)에서 모든 분기가 실행되었는지를 검사한다. if 문에서는 else-if와 else를 포함한 모든 분기의 실행 여부를, case 문에

```
1: always @(posedge Clk)begin
2:    if(OpClr)begin
3:       WrAddr = 5'd0; RdAddr = 5'd0;
4:    end
5:    else if(OpWr)begin
6:       Mem[WrAddr] = WrData; WrAddr = WrAddr + 1;
7:    end
8:    else if(OpRd)begin
9:       RdData = Mem[RdAddr];RdAddr = RdAddr + 1;
10:   end
11: end
```

그림 2. Branch coverage 예제
Fig. 2. Branch coverage example.

```
1: if(WrEn && !Full)
2: begin
3:    OpWr = 1'b1;
4:    NextState = ST_END
5: end
```

그림 3. Condition coverage 예제
Fig. 3. Condition coverage example.

서는 default를 포함한 모든 경우를 확인한다[7]. 이는 분기점의 모든 경로가 테스트 되었는지를 확인함으로써 코드의 각 분기 경로가 올바르게 작동하는지 보장한다. 예를 들어, 그림 2에서 2, 5, 8번째 줄 분기의 실행 여부는 branch coverage에서 커버해야 하는 목록으로써 나타난다. 이러한 목록의 충족 여부는 잠재적으로 미처리된 경로를 식별할 수 있게 하여 코드가 모든 예상 가능한 조건에 대응할 수 있도록 한다.

3. Condition Coverage

Condition coverage는 조건문(if 구문)에서 모든 조건이 실행되었는지를 검증한다[8]. Branch coverage는 if 문의 분기의 실행 여부만을 검증하는 반면, condition coverage는 if 문 내 조건의 논리적 조합의 테스트 여부를 검증한다. 이는 복잡한 논리적 조건이 올바르게 작동하는지를 보장한다. 예를 들어, 그림 3에서 1번째 줄의 if 문 내 조건이 모든 경우의 수(0/0, 0/1, 1/0, 1/1) 실행 여부가 condition coverage에서 커버해야 하는 목록으로써 나타난다. 이러한 목록의 충족 여부를 통해 복잡한 조건의 로직 내 오류를 효과적으로 발견할 수 있다[9].

4. Toggle Coverage

Toggle coverage는 코드 내 모든 신호가 가능한 모든 값 사이를 전환했는지를 검증한다. 신호의 모든 가능한 상태 변경을 추적함으로써, 코드가 다양한 입력 조건 하에서 안정적으로 작동하는지를 보장한다. 예를 들어, 그림 4의 모든 변수의 값 전환 여부가 toggle coverage에서 커버해야 하는 목록으로써 나타난다. 이

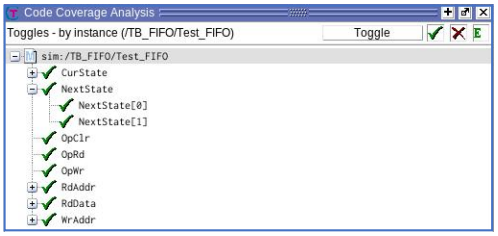


그림 4. Toggle coverage 예제
Fig. 4. Toggle coverage example.

```
1: assign Empty =((WrAddr-RdAddr)== 5'b00000)?1'b1:1'b0;
2: assign Full  =((WrAddr-RdAddr)== 5'b11111)?1'b1:1'b0;
```

그림 5. Expression coverage 예제
Fig. 5. Expression coverage example.

```
1:  always @(posedge Clk)
2:  begin OpWr = 1'b0; OpRd = 1'b0; OpClr = 1'b0;
3:     case(CurState)
4:        ST_IDLE: begin
5:                    OpClr = 1'b1;
6:                    NextState = ST_OP;
7:                 end
8:        ST_OP :  begin
9:                    if(WrEn && !Full)
10:                   begin
11:                      OpWr = 1'b1;
12:                      NextState = ST_END;
13:                   end
14:                   if(RdEn && !Empty)
15:                   begin
16:                      OpRd = 1'b1;
17:                      NextState = ST_END;
18:                   end
19:                end
20:       ST_END : begin
21:                   NextState = ST_OP;
22:                end
23:    endcase
24: end
```

그림 6. FSM coverage 예제
Fig. 6. FSM coverage example.

러한 목록의 충족 여부는 실제 하드웨어 환경에서 예상치 못한 문제를 일으킬 수 있는 미흡한 코드 부분을 식별할 수 있게 한다.

5. Expression Coverage

Expression coverage는 할당문(assign) 내에서 모든 표현이 실행되었는지 검사한다. 특히 복잡한 논리식에서 예상치 못한 오류를 찾아내는 데 큰 도움이 된다. 이는 할당된 각 신호에 대한 모든 가능한 계산이 수행되었는지를 보장한다. 예를 들어, 그림 5에서 1, 2번째 줄의 assign 문 내 조건의 충족 여부가 expression coverage에서 커버해야 하는 목록으로써 나타난다. 많은 리소스를 요구할 수 있지만[10], 이러한 목록의 충족 여부를 통해 검증 과정에서 발생할 수 있는 논리적 오류를 확인할 수 있다.

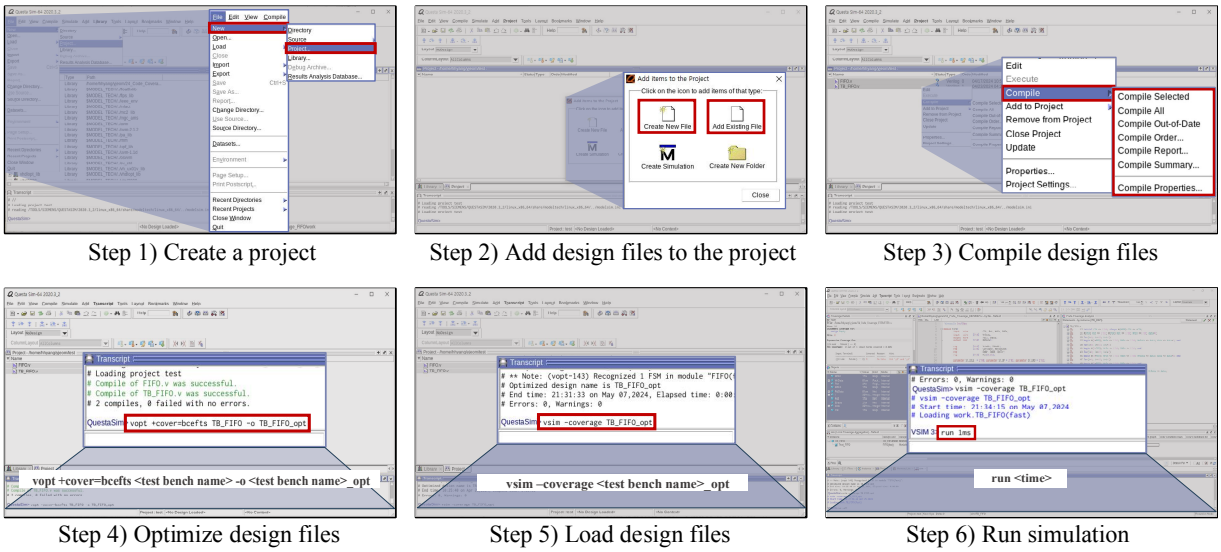


그림 7. Questa 시뮬레이션 흐름
Fig. 7. Simulation flow in Questa.

6. FSM Coverage

FSM (Finite State Machine) coverage는 FSM에서 모든 상태와 상태 간 전환을 실행했는지 검사한다. 이는 FSM 기반 로직이 안정적으로 동작하는지를 보장한다. 예를 들어, 그림 6에서 각 state의 실행 여부와 가능한 모든 transition의 수행 여부가 커버해야 하는 목록으로써 나타난다. 이러한 목록의 충족 여부를 통해 하드웨어 설계의 기능적 정확성을 보장할 수 있으며[11], FSM을 기반으로 하는 설계의 필수적인 검증 유형으로 간주된다.

설계자는 Mentor 사의 Questa를 사용하여 이러한 다양한 커버리지 유형들을 통해 코드의 각 부분이 제대로 실행되고 있는지, 모든 조건이 충족되었는지, 그리고 잠재적인 오류가 없는지를 체계적으로 검토할 수 있다. 또한 조건에 따라 커버해야 하는 목록이 생성되며. 커버 여부에 따라 달성도가 체크된다. 이 과정을 통해, 더욱 견고하고 오류 없는 설계를 구현할 수 있다.

Ⅲ. 본 론

본 논문에서는 두 가지 예제를 통해 code coverage 검증 절차에 대하여 설명한다. 우선, Mentor 사의 Questa를 활용한 code coverage 절차에 대하여 설명한다. 이후, 본 논문의 첫 번째 예제인 counter의 블록 다이어그램을 먼저 설명하고, FIFO의 블록 다이어그램과 FSM 구조를 설명한다.

1. Code Coverage 검증 방법

그림 7은 Questa를 사용하여 code coverage 검증 수행 과정을 여섯 단계로 세분화하여 설명한다. 첫 번째 단계에서는 Questa의 사용자 인터페이스를 통해 새 프로젝트를 시작한다. 이를 위해 메뉴에서 File > New > Project를 선택하여 새 프로젝트 창을 열고 필요한 프로젝트 설정을 입력한다. 두 번째 단계에서는 추가할 RTL 설계 파일을 프로젝트에 포함시키는 작업을 진행한다. Add to the project > Create New Files 또는 Existing Files 옵션을 사용해 필요한 설계 파일들을 추가하며, 이는 시뮬레이션을 위한 RTL 코드와 테스트 벤치를 포함한다. 세 번째 단계는 추가된 RTL 코드의 문법적 정확성을 검증하기 위해 컴파일을 수행하는 과정이다. Compile > Compile Selected 또는 All 옵션을 통해 선택한 파일 또는 프로젝트 내 모든 파일을 컴파일한다. 이 과정은 코드 내의 문법적 오류를 발견하고 수정할 기회를 제공하며, 후속 시뮬레이션 단계에서 발생할 수 있는 잠재적 문제를 사전에 방지한다. 네 번째 단계에서는 실제 code coverage를 수행하기 위한 준비를 한다. Transcript 창에 [vopt +cover=[bcefts] <module name> -o <module name>_opt] 명령어를 입력하여[12] 필요한 code coverage 옵션을 최적화하고 모듈을 구성한다. 여기서 [bcefts] 옵션을 통해 branch, condition, expression, FSM, toggle, statement 등의 coverage 옵션을 설정한다. 다섯 번째 단계에서는 Transcript 창에 [vsim -coverage <module name>_opt] 명령어를 입력하여 최적화된 모

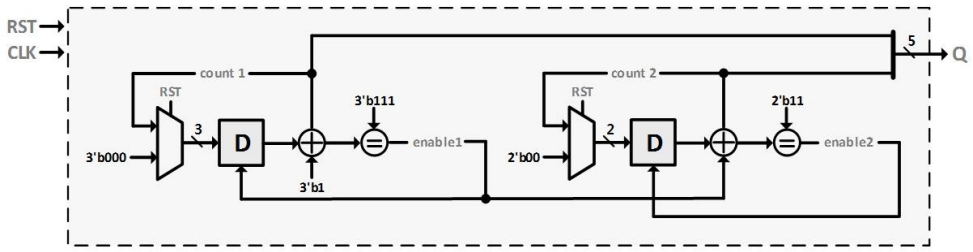


그림 8. Counter 블록 다이어그램
Fig. 8. Counter block diagram.

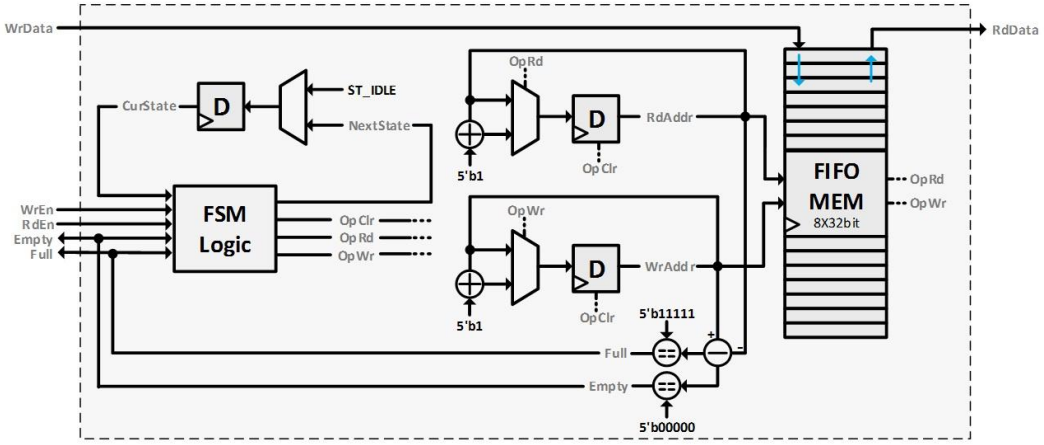


그림 9. FIFO 블록 다이어그램
Fig. 9. FIFO block diagram.

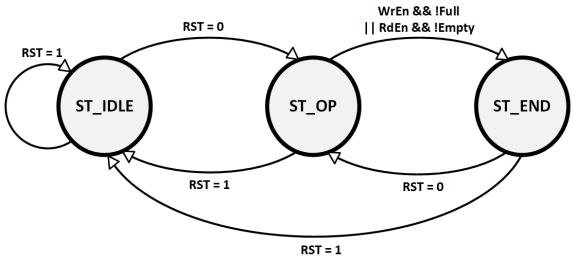


그림 10. FIFO 유한 상태 기계
Fig. 10. FIFO FSM.

듈을 불러온다. 이는 시뮬레이션 환경을 구성하고 준비된 설정에 따라 시뮬레이션을 시작할 준비를 마치는 단계이다. 여섯 번째 단계에서는 [run <time>] 명령을 사용하여 설정한 시간 동안 시뮬레이션을 실행한다. 이 시간은 시뮬레이션의 범위와 깊이를 결정하며, 충분한 시간 동안 시뮬레이션을 진행함으로써 코드의 모든 측면이 제대로 검증될 수 있도록 한다. 마지막으로 시뮬레이션 완료 후 Tools > Coverage Report > Text or HTML 옵션을 통해 code coverage 결과를 확인할 수 있다.

표 2. Questa의 code coverage 아이콘
Table 2. Code coverage icons in Questa.

Icon	Description
✓	모든 coverage를 충족함
X	2개 이상의 coverage 유형을 충족하지 못함
X_T	Branch coverage가 참일 조건을 만족하지 못함
X_F	Branch coverage가 거짓일 조건을 만족하지 못함
X_C	Condition coverage를 충족하지 못함
X_E	Expression coverage를 충족하지 못함
X_B	Branch coverage를 충족하지 못함
X_S	Statement coverage를 충족하지 못함

또한, Questa는 시뮬레이션 동안 충족되지 않은 code coverage에 대한 정보를 그래픽으로 제공하며, 표 2에서는 이 정보를 나타내는 아이콘의 의미를 정리한다.

```
1: module Counter(
2:    input  wire        Clk, Rst,
3:    output wire  [4:0] Q
4: );
5:
6:    reg  [2:0] count1;
7:    reg  [1:0] count2;
8:   wire        enable1;
9:   wire      enable2;
10:
11:    assign enable1 = (count1 == 3'b111) ? 1'b1 : 1'b0;
12:    assign enable2 = (count2 == 3'b111) ? 1'b1 : 1'b0;
13:
14:    always @(posedge Clk or posedge Rst) begin
15:        if (Rst) begin
16:            count1 <= 3'b000;
17:        end else begin
18:            if (enable1) begin
19:                count1 <= 3'b000;
20:            end else begin
21:                count1 <= count1 + 1;
22:            end
23:        end
24:    end
25:
26:    always @(posedge Clk or posedge Rst) begin
27:        if (Rst) begin
28:            count2 <= 2'b00;
29:        end else if (enable1) begin
30:            if (enable2) begin
31:                count2 <= 2'b00;
32:            end else begin
33:                count2 <= count2 + 1;
34:            end
35:        end
36:    end
37:
38:    assign Q = {count2, count1};
39:
40: endmodule
```

그림 11. 수정 전 counter RTL 코드
Fig. 11. Erroneous counter RTL code.

```
1: module Counter(
2:    input  wire        Clk, Rst,
3:    output wire  [4:0] Q
4: );
5:
6:    reg  [2:0] count1;
7:    reg  [1:0] count2;
8:   wire        enable1;
9:   wire      enable2;
10:
11:    assign enable1 = (count1 == 3'b111) ? 1'b1 : 1'b0;
12:    assign enable2 = (count2 == 2'b11) ? 1'b1 : 1'b0;
13:
14:    always @(posedge Clk or posedge Rst) begin
15:        if (Rst) begin
16:            count1 <= 3'b000;
17:        end else begin
18:            if (enable1) begin
19:                count1 <= 3'b000;
20:            end else begin
21:                count1 <= count1 + 1;
22:            end
23:        end
24:    end
25:
26:    always @(posedge Clk or posedge Rst) begin
27:        if (Rst) begin
28:            count2 <= 2'b00;
29:        end else if (enable1) begin
30:            if (enable2) begin
31:                count2 <= 2'b00;
32:            end else begin
33:                count2 <= count2 + 1;
34:            end
35:        end
36:    end
37:
38:    assign Q = {count2, count1};
39:
40: endmodule
```

그림 13. 수정 후 counter RTL 코드
Fig. 13. Modified counter RTL code.

report.txt

```
Coverage Report Summary Data by file

================================================================================
=== File: /home/hhyang/yjeom/24_Code_Coverage_FIFO/COUNTER_error.v
================================================================================
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                        9         8         1    88.88%
    Expressions                     2         1         1    50.00%
    Statements                     11        10         1    90.90%
    Toggles                        14        12         2    85.71%
```

그림 12. 수정 전 code coverage 결과
Fig. 12. Code coverage result of erroneous FIFO RTL code.

report.txt

```
Coverage Report Summary Data by file

================================================================================
=== File: /home/hhyang/yjeom/24_Code_Coverage_FIFO/COUNTER.v
================================================================================
    Enabled Coverage             Bins      Hits    Misses  Coverage
    ----------------             ----      ----    ------  --------
    Branches                        9         9         0   100.00%
    Expressions                     2         2         0   100.00%
    Statements                     11        11         0   100.00%
    Toggles                        14        14         0   100.00%
```

그림 14. 수정 후 code coverage 결과
Fig. 14. Code coverage result of modified FIFO RTL code.

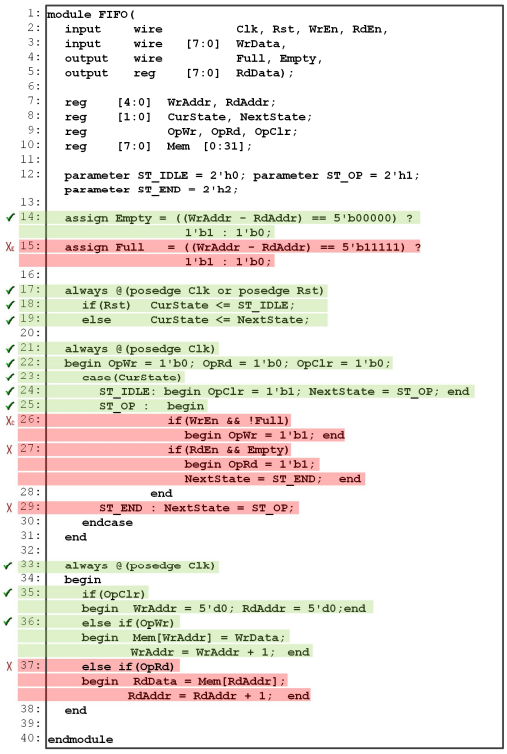
2. Counter

본 논문에서 다루는 DUT(Design Under Test)인 counter 구조는 여러 개의 카운터를 직렬으로 연결해 큰 범위를 카운트하는 구조이다. 해당 구조는 모듈화가 용이해 설계와 디버깅이 상대적으로 쉽다. 설계한 counter는 그림 8과 같이 시스템 입력으로 클록 신호(Clock, CLK)와 리셋 신호(Reset, RST)를 받으며, 카운트된 수인 Q를 출력한다.

3비트 counter와 2비트 counter를 직렬로 연결하여 5bit 크기의 counter를 구현하며 각 counter는 flip-flop으로 구성된다. 3비트 counter의 출력이 2비트 counter의 입력 클럭으로 연결되며, 3비트 counter가 최대 값에 도달하면 2비트 counter가 증가하게 되는 방식으로 동작한다.

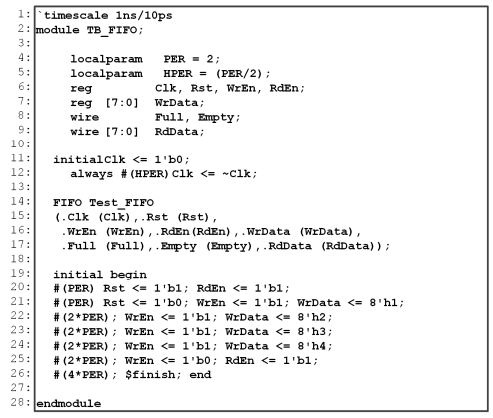
3. FIFO(First-In, First-Out)

본 논문에서 다루는 DUT(Design Under Test)인 FIFO (First-In, First-Out) 구조는 데이터가 입력된 순서대로 처리되는 메모리 구조다. 설계한 FIFO는 그림 9와 같이 시스템 입력으로 클록 신호(Clock, CLK)와 리셋 신호(Reset, RST)가 있고, 제어 입력으로는 쓰기를 가능하게 하는 WrEn (Write Enable)과 읽기를 가능하게 하는 RdEn(Read Enable)이 있다. 데이터는 8비트의 WrData로 입력되며, 8비트의 RdData로 출력된다. 또한 내부 FIFO 상태 신호인 Full과 Empty를 출력한다. 그림 9는 FIFO의 내부 블록 다이어그램을 자세히 나타내고 있다. WrEn 또는 RdEn 신호를 입력받아 내부적으로 FIFO가 가득 찼는지(Full), 비어 있는지(Empty)를 판단하고, 이에 따라 데이터 쓰기 또는 읽기



```
1:  module FIFO(
2:    input     wire          Clk, Rst, WrEn, RdEn,
3:    input     wire   [7:0]  WrData,
4:    output    wire          Full, Empty,
5:    output    reg    [7:0]  RdData);
6:
7:    reg     [4:0]  WrAddr, RdAddr;
8:    reg     [1:0]  CurState, NextState;
9:    reg            OpWr, OpRd, OpClr;
10:   reg     [7:0]  Mem  [0:31];
11:
12:   parameter ST_IDLE = 2'h0; parameter ST_OP = 2'h1;
      parameter ST_END = 2'h2;
13:
14:   assign Empty = ((WrAddr - RdAddr) == 5'b00000) ?
                     1'b1 : 1'b0;
15:   assign Full   = ((WrAddr - RdAddr) == 5'b11111) ?
                     1'b1 : 1'b0;
16:
17:   always @(posedge Clk or posedge Rst)
18:     if(Rst)   CurState <= ST_IDLE;
19:     else      CurState <= NextState;
20:
21:   always @(posedge Clk)
22:   begin OpWr = 1'b0; OpRd = 1'b0; OpClr = 1'b0;
23:     case(CurState)
24:       ST_IDLE: begin OpClr = 1'b1; NextState = ST_OP; end
25:       ST_OP :  begin
26:                  if(WrEn && !Full)
                     begin OpWr = 1'b1; end
27:                if(RdEn && Empty)
                     begin OpRd = 1'b1;
                     NextState = ST_END;  end
28:              end
29:       ST_END : NextState = ST_OP;
30:     endcase
31:   end
32:
33:   always @(posedge Clk)
34:   begin
35:     if(OpClr)
        begin  WrAddr = 5'd0; RdAddr = 5'd0;end
36:     else if(OpWr)
        begin  Mem[WrAddr] = WrData;
               WrAddr = WrAddr + 1;  end
37:     else if(OpRd)
        begin  RdData = Mem[RdAddr];
               RdAddr = RdAddr + 1;  end
38:   end
39:
40: endmodule
```

(a)



```
1:  `timescale 1ns/10ps
2:  module TB_FIFO;
3:
4:      localparam   PER = 2;
5:      localparam   HPER = (PER/2);
6:      reg          Clk, Rst, WrEn, RdEn;
7:      reg  [7:0]   WrData;
8:      wire         Full, Empty;
9:      wire [7:0]   RdData;
10:
11:   initialClk <= 1'b0;
12:     always #(HPER)Clk <= ~Clk;
13:
14:   FIFO Test_FIFO
15:   (.Clk (Clk),.Rst (Rst),
16:    .WrEn (WrEn),.RdEn(RdEn),.WrData (WrData),
17:    .Full (Full),.Empty (Empty),.RdData (RdData));
18:
19:   initial begin
20:   #(PER) Rst <= 1'b1; RdEn <= 1'b1;
21:   #(PER) Rst <= 1'b0; WrEn <= 1'b1; WrData <= 8'h1;
22:   #(2*PER); WrEn <= 1'b1; WrData <= 8'h2;
23:   #(2*PER); WrEn <= 1'b1; WrData <= 8'h3;
24:   #(2*PER); WrEn <= 1'b1; WrData <= 8'h4;
25:   #(2*PER); WrEn <= 1'b0; RdEn <= 1'b1;
26:   #(4*PER); $finish; end
27:
28: endmodule
```

(b)

그림 15. (a) 수정 전 FIFO RTL 코드; (b) 수정 전 FIFO 테스트 벤치

Fig. 15. (a) Erroneous FIFO RTL code; (b) Erroneous FIFO test bench.

report.txt

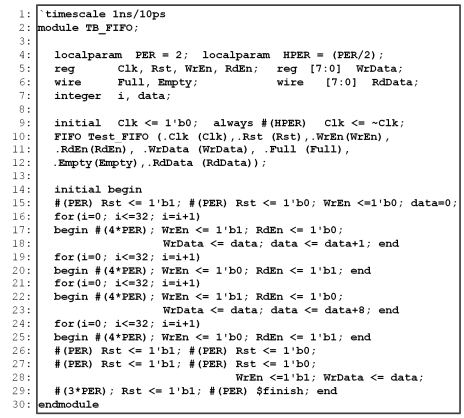
```
Coverage Report Summary Data by file

================================================================================
=== File: /home/hhyang/yjeom/24_Code_Coverage_FIFO/FIFO_error.v
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        14        11         3    78.57%
    Conditions                       4         1         3    25.00%
    Expressions                      2         1         1    50.00%
    FSM States                       3         2         1    66.66%
    FSM Transitions                  5         1         4    20.00%
    Statements                      22        17         5    77.27%
    Toggles                         50        10        40    20.00%
```

그림 16. 수정 전 code coverage 결과

Fig. 16. Code coverage result of erroneous FIFO RTL code.


```
1:  module FIFO(
2:    input     wire        Clk, Rst, WrEn, RdEn,
3:    input     wire [7:0]  WrData,
4:    output    wire        Full, Empty,
5:    output    reg  [7:0]     RdData);
6:
7:    reg     [4:0]  WrAddr, RdAddr;
8:    reg     [1:0]  CurState, NextState;
9:    reg            OpWr, OpRd, OpClr;
10:   reg     [7:0]  Mem[0:31];
11:
12:   parameter ST_IDLE = 2'h0; parameter ST_OP = 2'h1;
      parameter ST_END = 2'h2;
13:
14:   assign Empty  = ((WrAddr - RdAddr) == 5'b00000) ?
                      1'b1 : 1'b0;
15:   assign Full   = ((WrAddr - RdAddr) == 5'b11111) ?
                      1'b1 : 1'b0;
16:
17:   always @(posedge Clk or posedge Rst)
18:     if(Rst)   CurState <= ST_IDLE;
19:     else      CurState <= NextState;
20:
21:   always @(posedge Clk)
22:   begin OpWr = 1'b0; OpRd = 1'b0; OpClr = 1'b0;
23:     case(CurState)
24:       ST_IDLE: begin OpClr = 1'b1; NextState = ST_OP; end
25:       ST_OP :  begin
26:                  if(WrEn && !Full)
                     begin OpWr = 1'b1;
                     NextState = ST_END; end
27:                if(RdEn && !Empty)
                     begin OpRd = 1'b1;
                     NextState = ST_END; end
28:              end
29:       ST_END : NextState = ST_OP;
30:     endcase
31:   end
32:
33:   always @(posedge Clk)
34:   begin
35:     if(OpClr)
        begin  WrAddr = 5'd0; RdAddr = 5'd0; end
36:     else if(OpWr)
        begin  Mem[WrAddr] = WrData;
               WrAddr = WrAddr + 1;  end
37:     else if(OpRd)
        begin  RdData = Mem[RdAddr];
               RdAddr = RdAddr + 1;  end
38:   end
39:
40: endmodule
```

(a)



```
1:  `timescale 1ns/10ps
2:  module TB_FIFO;
3:
4:    localparam  PER = 2;  localparam  HPER = (PER/2);
5:    reg       Clk, Rst, WrEn, RdEn;   reg  [7:0]  WrData;
6:    wire      Full, Empty;            wire   [7:0]  RdData;
7:    integer   i, data;
8:
9:    initial   Clk <= 1'b0;  always #(HPER)   Clk <= ~Clk;
10:   FIFO Test_FIFO (.Clk (Clk),.Rst (Rst),.WrEn(WrEn),
11:   .RdEn(RdEn), .WrData (WrData), .Full (Full),
12:   .Empty(Empty),.RdData (RdData));
13:
14:   initial begin
15:   #(PER) Rst <= 1'b1; #(PER) Rst <= 1'b0; WrEn <=1'b0; data=0;
16:   for(i=0; i<=32; i=i+1)
17:   begin #(4*PER); WrEn <= 1'b1; RdEn <= 1'b0;
18:                   WrData <= data; data <= data+1; end
19:   for(i=0; i<=32; i=i+1)
20:   begin #(4*PER); WrEn <= 1'b0; RdEn <= 1'b1; end
21:   for(i=0; i<=32; i=i+1)
22:   begin #(4*PER); WrEn <= 1'b1; RdEn <= 1'b0;
23:                   WrData <= data; data <= data+8; end
24:   for(i=0; i<=32; i=i+1)
25:   begin #(4*PER); WrEn <= 1'b0; RdEn <= 1'b1; end
26:   #(PER) Rst <= 1'b1; #(PER) Rst <= 1'b0;
27:   #(PER) Rst <= 1'b1; #(PER) Rst <= 1'b0;
28:                              WrEn <=1'b1; WrData <= data;
29:   #(3*PER); Rst <= 1'b1; #(PER) $finish; end
30: endmodule
```

(b)

그림 17. (a) 수정 전 FIFO RTL 코드; (b) 수정 전 FIFO 테스트 벤치

Fig. 17. (a) Erroneous FIFO RTL code; (b) Erroneous FIFO test bench.

report.txt

```
Coverage Report Summary Data by file

================================================================================
=== File: /home/hhyang/yjeom/24_Code_Coverage_FIFO/FIFO_error.v
================================================================================
    Enabled Coverage              Bins      Hits    Misses  Coverage
    ----------------              ----      ----    ------  --------
    Branches                        14        11         3    78.57%
    Conditions                       4         1         3    25.00%
    Expressions                      2         1         1    50.00%
    FSM States                       3         2         1    66.66%
    FSM Transitions                  5         1         4    20.00%
    Statements                      22        17         5    77.27%
    Toggles                         50        10        40    20.00%
```

그림 18. 수정 전 code coverage 결과

Fig. 18. Code coverage result of erroneous FIFO RTL code.

동작을 제어한다. Write Logic 부분은 활성화된 WrEn 신호를 기반으로 WrData를 현재 WrAddr로 지정된 메모리 주소에 저장하고, 데이터가 성공적으로 기록된 후 WrAddr 값을 증가시켜 다음 데이터를 위한 공간을 준비한다. Read Logic은 활성화된 RdEn 신호 하에 RdAddr에서 데이터를 읽어와 RdData로 출력하고, 읽기 작업 후 RdAddr을 증가시켜 다음 읽어올 데이터에 대한 주소를 준비한다.

그림 10과 같이 FIFO의 제어는 FSM (Finite State Machine)에 의해 관리되며, 이는 세 가지 주요 상태로 구성된다. 초기 상태인 IDLE에서는 RST 신호가 활성화되어 있을 때 시스템이 대기 상태를 유지한다. RST 신호가 비활성화되면 시스템은 OP (Operation) 상태로 전환하여, WrEn 또는 RdEn 신호에 의해 트리거된 적절한 쓰기 또는 읽기 작업을 시작한다. 각 작업은 Full과 Empty 상태를 검토하여 작업 가능 여부를 결정한다. 작업이 완료되면 시스템은 END 상태로 이동하여 모든 작업이 마무리되고, 다시 OP 상태로 돌아가 다음 동작을 대기한다. 이 FSM 구조는 FIFO가 효율적으로 데이터를 순차적으로 처리하도록 지원하며, 다양한 데이터 스트림 처리 상황에서 FIFO의 안정성과 신뢰성을 보장한다.

Ⅳ. 실험 결과

본 논문에서는 오류를 포함하는 counter와 FIFO RTL code를 예제로 하여 code coverage 검증을 진행하였다. 우선, 본 논문의 첫 번째 예제인 counter의 code coverage 검증 결과를 통해 오류 코드를 수정하고, 이후 두 번째 예제인 FIFO의 검증 결과를 통해 오류 코드를 수정했다.

1. Counter 검증 결과

그림 8에 제시된 counter를 code coverage 검증 절차 설명을 위해 의도적으로 오류를 포함하도록 설계했다. 그림 11은 오류를 포함한 RTL 코드를 보여주고 있으며, 주어진 오류를 포함한 코드의 검증을 그림 7의 절차에 따라 진행하였다. 검증 결과는 그림 12에서 확인 가능하다. 그림 12의 Bins는 전체 케이스를 의미하며, 성공을 나타내는 Hits와 실패를 나타내는 Misses를 통해 각 coverage의 비율이 제시된다. 구체적으로 그림 12의 결과에 따르면, branch coverage는 88.88%, expressions coverage는 50.00%, statements coverage는 90.90%, 그리고 toggle coverage는 85.71%를 기록했다. 이러한 데이터를 통해 그림 11의 RTL 코드에서 나타난 모든 coverage 유형을 충족하지 못했음을 확인할 수 있다. 예를 들어, 표 2와 그림 12의 분석에 따르면, 12줄의 아이콘은 expression coverage가 충족되지 않음을 의미한다. 이를 통해 count2 신호가 3'b111인 케이스를 충족하지 못했으며, 이는 count2 신호가 2비트로 선언되어 비트 크기가 맞지 않아 충족되지 못했음을 알 수 있다. 30줄과 31줄에서도 enable2 신호가 1인 케이스를 충족하지 못해 branch, statement coverage를 충족하지 못했음을 보여준다.

그림 13은 이러한 분석을 바탕으로 수정된 RTL 코드를 보여주고 있으며, 그림 14는 이에 대한 code coverage 결과를 보인다. 수정 후의 결과에서는 나타난 모든 coverage 유형이 충족되었음을 확인할 수 있으며, 이를 통해 code coverage를 기반으로 한 검증에서 의도한 모든 오류를 발견했음을 확인 가능하다.

2. FIFO 검증 결과

그림 9에 제시된 FIFO를 code coverage 검증 절차 설명을 위해 의도적으로 오류를 포함하도록 설계했다. 그림 15는 오류를 포함한 RTL 코드와 테스트 벤치를 보여주고 있으며, 테스트 벤치는 의도적으로 테스트 케이스가 부족하게 구성되었다. 주어진 오류를 포함한 코드의 검증을 그림 7의 절차에 따라 진행하였다. 검증 결과는 그림 16에서 확인 가능하며, branch coverage는 78.57%, conditions coverage는 25.00%, expressions coverage는 50.00%, FSM의 state coverage는 66.66% 및 transition coverage는 20.00%, statements coverage는 77.27%, 그리고 toggle coverage는 20.00%를 기록했다. 이러한 데이터를 통해 그림 15의 RTL 코드가 모든 coverage 유형을 충족하지 못했음을 확인할 수 있다. 표 2과 그림 15의 분석에 따르면, 15줄의 아이콘은 expression coverage가 충족되지 않음을 의미하며, 이를 통해 Full 신호가 1인 케이스를 충족하지 못했음을 알 수 있다. 26줄의 아이콘은 write 동작의 조건이 충족되어 OP 상태에서 구문이 실행되었으나, 마찬가지로 Full 신호가 1인 케이스를 충족하지 못해 condition coverage가 충족되지 않았음을 보여준다. 27줄에서는 2가지 이상의 coverage 유형이 충족되지 않았으며, 이는 read 동작의 조건이 잘못 기술되어 있어 구문이 실행되지 않았기

때문임을 알 수 있다. 29줄에서도 모든 coverage 유형이 충족되지 않아, END state로의 전환 실패가 확인되었고, 이에 따라 전환을 위한 구문이 누락되었음을 파악할 수 있다.

그림 17은 이러한 분석을 바탕으로 수정된 RTL 코드와 테스트 벤치를 보여주고 있으며, 그림 18은 이에 대한 code coverage 결과를 보인다. 수정 후의 결과에서는 모든 coverage 유형이 충족되었음을 확인할 수 있으며, 이를 통해 code coverage를 기반으로 한 검증에서 의도한 모든 오류를 발견했음을 확인 가능하다.

본 논문에서는 단순한 구조인 counter와 FIFO를 예로 들었지만, 실무적으로는 더 복잡한 회로에 대해 동일한 방법으로 검증을 진행할 수 있다. Code coverage를 통해 모든 코드가 적절히 테스트 되었는지, 모든 조건이 충족되었는지, 잠재적인 오류가 없는지를 체계적으로 검토할 수 있다. 이러한 과정은 더욱 견고하고 오류 없는 설계를 구현하는 데 기여할 수 있다.

V. 결 론

본 논문에서는 Questa를 활용한 code coverage 검증 절차를 소개한다. 특히, counter와 FIFO RTL 코드를 사용한 예시와 수정 과정을 통해, 각 코드의 coverage 유형별 충족 여부를 확인하고 의도된 설계 오류 및 테스트 케이스 부족 오류를 식별하였다. 이 과정은 code coverage 검증이 RTL 코드 검증에 있어 매우 효율적인 방법임을 입증하는 결과를 제공했다.

Code coverage 검증은 다양한 분야에 적용 가능하며, 특히 항공 기술 분야에서는 높은 신뢰성과 무결성을 보장하기 위해 필수적이다. 최근 항공 소프트웨어 표준인 DO-178이 항공 전자 하드웨어의 신뢰성을 보장하는 DO-254 표준으로 확장되면서, code coverage 분석은 모든 상태 및 조건을 검증하고 테스트의 완결성 및 효과성을 평가하는 중요한 도구로 자리잡았다. 이러한 추세는 앞으로도 계속될 것으로 예상되며, code coverage는 다양한 분야에서 설계 오류의 조기 발견과 해결을 가능하게 함으로써 전체 시스템의 안정성과 신뢰성을 크게 향상시킬 것이다.

REFERENCES

[1] Y. Akhilesh, J. Poonam, M. Fyrbiak, B. Devaraju, "A Survey on Assertion-based Hardware Verification," in *ACM Computing Surveys*, Vol. 54, No. 11s, Article 225, September. 2022.

[2] Harry D. Foster, “Trends in Functional Verification: A 2014 Industry Study,” in *DAC '15: Proceedings of the 52nd Annual Design Automation Conference,* Jun. 2015.

[3] W. Hasini, L. Yangdi, C. Subodha, M. Prabhat, "Study And Analysis of RTL Verification Toop" in *2020 IEEE Students Conference on Engineering & Systems (SCES)*, July. 2020.

[4] E. Hatem, K. Mostafa, S. Amr, K. Mohammed, " A Novel Assertions-Based Code Coverage Automatic CAD Tool," in *IEEE EUROCON 2017 -17th International Conference on Smart Technologies,* July 2017.

[5] A. Viraj, M. Sai, H. Samuel, V. Shobha, “Code Coverage of Assertions Using RTL Source Code Analysis,” in *DAC '14: Proceedings of the 51st Annual Design Automation Conference*, June 2014.

[6] G. Alon, “Practical Methods in Coverage-Oriented Verification of the Merom Microprocessor,” in *DAC '06: Proceedings of the 43rd annual Design Automation Conference*, July 2006.

[7] Vineeth V. Acharya, Sharad Bagri and Michael S. Hsiao, “Branch Guided Functional Test Generation at the RTL,” in *2015 20th IEEE European Test Symposium (ETS),* July 2015.

[8] John Sanguinetti, Eugene Zhang, “The Relationship of Code Coverage Metrics on High-level and RTL Code,” in *2010 IEEE International High Level Design Validation and Test Workshop (HLDVT),* June 2010.

[9] Yuan-Bin Sha, Mu-Shun Lee and Chien-Nan Jimmy Liu, “On Code Coverage Measurement for Verilog-A,” in *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop*, November 2004.

[10] Graeme D, Paul B. Jackson, Julian A, “Expression Coverability Analysis: Improving code coverage with model checking,” in *DVCON'04,* 2004.

[11] W. Tsu-Hua, E. Thomas, “Practical FSM Analysis for Verilog,” in *Proceedings International Verilog HDL Conference and VHDL International Users Forum*, March 1998.

[12] Mentor Graphics Corporation, “Questa SIM User’s Manual”, 2011.

저 자 소 개

엄 유 진(학생회원)
2021년~현재 충남대학교
전자공학과 학사 과정

<주관심분야 : FPGA 플랫폼, RTL 검증>

양 희 훈(학생회원)
2020년 충남대학교
전자공학과 학사 졸업.
2023년~현재 충남대학교
전자공학과
석박사통합과정

<주관심분야 : FPGA 플랫폼, FPGA 난수발생기, FPGA 역공학>

심 용 기(비회원)
2012년 충남대학교
메카트로닉스공학과
학사 졸업.
2021년 LIG넥스원 항공연구소
선임 연구원.
2023년 한국항공우주산업
선행기술실 선임연구원.
2024년 충남대학교 전자공학과 석사 졸업.
<주관심분야 : 항공전자, Safety Critical FPGA 설계, 고신뢰도 설계 및 검증>

배 득 우(비회원)
2007년 경남대학교
정보통신공학부
학사 졸업.
2009년 경남대학교
정보통신공학과
석사 졸업.
2014년 부산대학교 전자전기공학과 박사 수료.
2008년~2015년 한국전기연구원 선임연구원.
2015년~현재 한국항공우주산업 책임연구원.
<주관심분야 : 위성 탑재컴퓨터, 항공기 비행제어 컴퓨터>

오 민 식(비회원)
2015년 창원대학교 전자공학과
학사 졸업.
2015년~2022년
DNKR(덴소코리아)
연구원.
2022년~현재 한국항공우주산업
선임연구원.
<주관심분야 : FPGA 플랫폼, 회로설계>

유 호 영(정회원)
2010년 연세대학교
전기전자공학부
학사 졸업.
2012년 KAIST 전자공학과
석사 졸업.
2016년 KAIST 전자공학과
박사 졸업.
2016년 삼성전자 메모리사업부 책임 연구원.
2020년 충남대학교 전자공학과 조교수.
2024년 충남대학교 전자공학과 부교수.
2024년~현재 충남대학교 전자공학과 정교수.
<주관심분야 : FPGA 플랫폼, FPGA 역공학, PIM, Error Correction Code>